Trabalho 2 - Processamento de Imagens

Décio Gonçalves de Aguiar Neto[1]

[1]Universidade Estadual de Campinas - UNICAMP
Instituto de Computação

1. IMPLEMENTAÇÃO

Para a implementação deste trabalho foi utilizada a biblioteca imageio para fazer o carregamento da imagem e salvar a imagem após a codificação.

Na implementação do codificador foi necessário o uso de funções extras que ajudassem na manipulação das camadas RGB assim como funções que pudessem fazer a alteração de bits específicos de cada pixel.

Um dos primeiros desafios relacionados a conversão dos caracteres em valores binário por exemplo foi o fato de que o valor em decimal para o caractere space na tabela ASCII quando convertido para binário pelo uso da função bin() nativa do python gera um binário com menos de 8bits, para manter um padrão de tamanho da palavra de bits uma função de conversão de caracteres foi implementada fazendo o ajuste preenchendo com 0's à esquerda as cadeias que não possuíssem esse tamanho, esse ajuste foi feito para evitar problemas futuras na função que extrai o texto codificado na imagem.

Conforme descrito na especificação do trabalho o algoritmo de codificação recebe a imagem original, a mensagem a ser ocultada dentro da imagem e o plano de bits onde será feita a alteração em cada pixel. A mensagem é transformada em uma cadeia de bits que é percorrida enquanto preenche a matriz. No final após toda a cadeia de bits ser percorrida a imagem é retornada.

O método de decodificar recebe como parâmetros conforme descrito na especificação apenas a imagem alterada e o plano de bits que será percorrido onde a mensagem foi ocultada, esta implementação não estabeleceu um critério de parada para a busca da mensagem, uma possível solução seria o uso de um algum caractere limitador que indicasse que toda a mensagem foi encontrada e encerrando o laço de busca na imagem. Quando a mensagem é recuperada pelo fato de percorrer toda a imagem o algoritmo também retorna lixo na sua saída ao escrever no arquivo.

Para a apresentação do plano de bits e para gerar os planos desejados é feito uma transformação na imagem, onde cada canal é binarizado para o valor de bit do plano que se deseja exibir. A forma utilizada para fazer essa transformação é descrita na Equação 1, onde M é a imagem de entrada e p é o plano o qual se deseja pegar informação, essa transformação garante que o resultado sempre seja binário. Um observação interessante foi que ao realizar essa transformação sobre imagens de extensão (*.jpg) e se tentava exibir os planos de bits específicos não foi possível observar os espectros da imagem, já que a imagem exibida era sempre preta, diferente das imagens com extensão (*.png) que se mostraram comportadas ao exibir os planos de bits desejados.

$$M = \lfloor \frac{M}{2^p} \rfloor \pmod 2 \tag{1}$$

2. EXECUÇÃO

Para que o código seja executado de forma correta basta executar os seguintes comandos:

```
python3 encode.py caminho_da_imagem text.txt plano_bits imagem_saida.png
```

```
python3 decode.py caminho_da_imagem plano_bits
```

A saída será dada da seguinte forma, para o método encode, a imagem codificada será retornada. e para o método decode, ele irá retornar um arquivo contendo a mensagem que estava dentro da imagem.

3. RESULTADOS

Nesta seção são apresentados os resultados obtidos passo-a-passo. A imagem original utilizada é a Figura 1, seguida de suas versões do plano $0, 1, 2$ e 7 de bits, a imagem foi alterada no plano 0, na Figura 2 podemos observar que na parte superior uma certa alteração no padrão dos pixels contidos na imagem, isso é o esperado já que o plano de bits modificado foi o plano 0. Já as outras Figuras exibidas não apresentam nenhuma anormalidade já que não houve alteração nesses outros planos de bit.

Figura 1. Imagem Original

Figura 2. Plano de bits 0

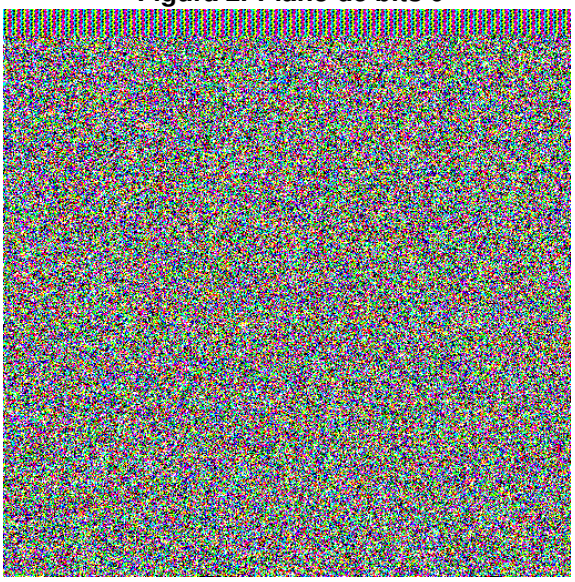

Figura 3. Plano de bits 1

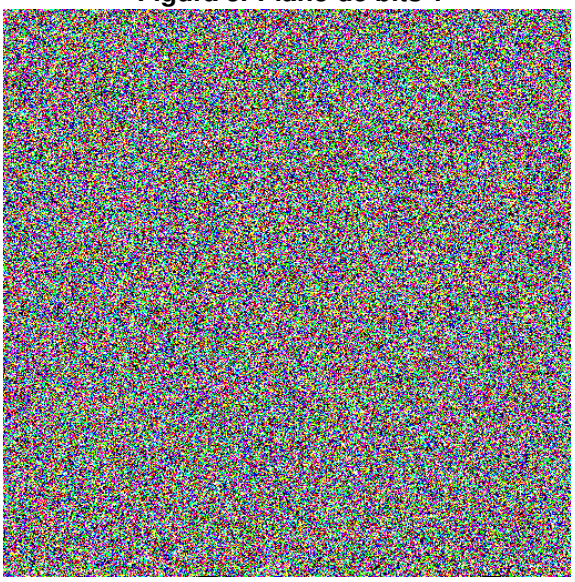

Figura 4. Plano de bits 2

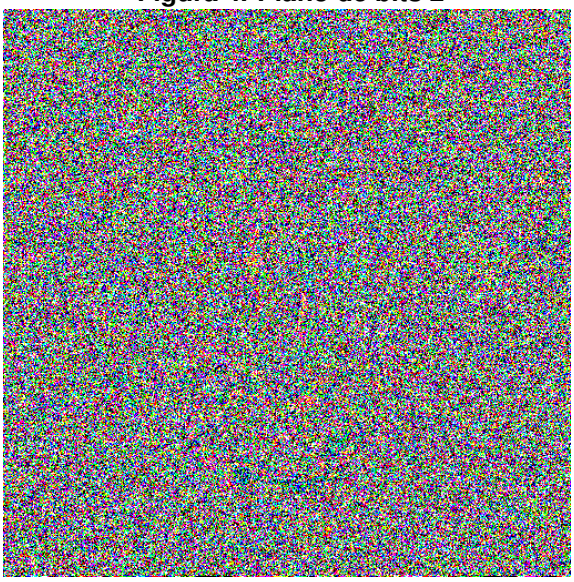

Figura 5. Plano de bits 7